



# Simple Digital Logic Design (H-Bridge)

Georgia Institute of Technology

CS 3651 – Prototyping  
Intelligent Appliances

# Simple Digital Logic Design

- The basic steps in designing a simple digital circuit are:
  - Step 1: Define the problem
    - Truth tables
  - Step 2: Translate truth tables into combinatorial logic circuit
    - Boolean Algebra
    - Minterms
    - Sum of Products (or Product of Sums)
  - Step 3: Optimization
    - Boolean Identities
    - DeMorgan's Law
    - Karnaugh Maps (K-Maps)
  - Step 4: Build It!
    - Protoboard and Integrated Circuits.
- Warning: This is a lot of information if it is your first exposure to circuits!

## Step 1: Define the Problem

- Digital logic circuits can contain multiple inputs and outputs.
- The combinations of inputs and outputs can be represented in a table form (called truth tables).
- Truth tables should list ALL the combinations of inputs and outputs.

Example: Inverter

Input	Output
0	1
1	0

## Our Problem: H-bridge

- We want to build a device called an H-Bridge.
  - An H-bridge is a simple motor controller that is used to provide 4 functions to an electric motor: Forward, Reverse, Brake, and Coast. The functions are selected with 2 input lines.
  - The H-bridge is built with 4 switches, and allows voltage to be applied across the motor in either direction.

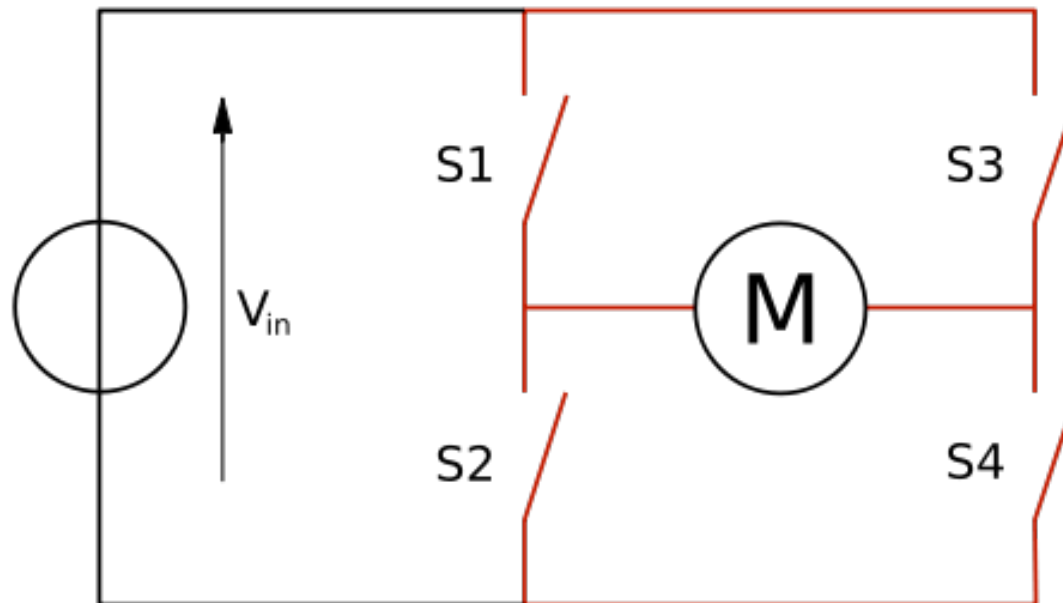


Photo courtesy of Wikipedia

# H-Bridge Truth Table

- H-Bridge input table      H-Bridge output table

IN 2	IN 1	Function
0	0	Coast
0	1	Forward
1	0	Reverse
1	1	Brake

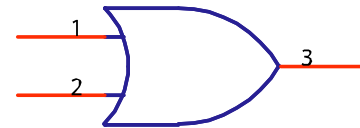
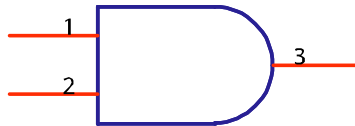
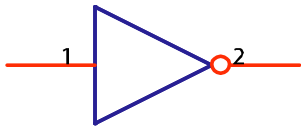
Function	SW1	SW2	SW3	SW4
Coast	0	0	0	0
Forward	1	0	0	1
Reverse	0	1	1	0
Brake	1	1	0	0

- H-Bridge full combinatorial logic full truth table

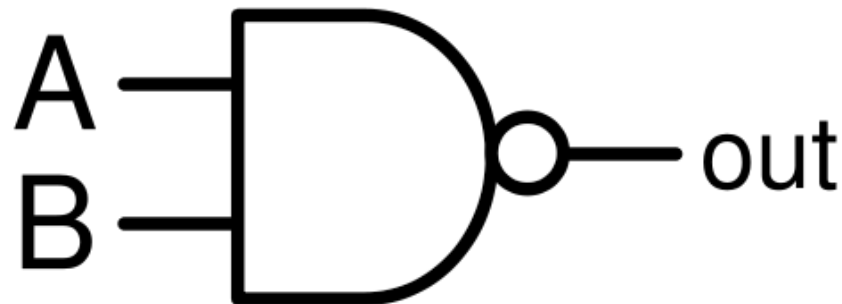
IN2	IN1	SW1	SW2	SW3	SW4	Function
0	0	0	0	0	0	Coast
0	1	1	0	0	1	Forward
1	0	0	1	1	0	Reverse
1	1	1	1	0	0	Brake

## Step 2: Translate truth table into circuit

- All digital circuits can be built with some combination of AND, OR, and NOT gates.
- Depending on what gates you have available, you can redesign your circuit to use different types of gates (using DeMorgan's Law).
- NAND gates are widely used.

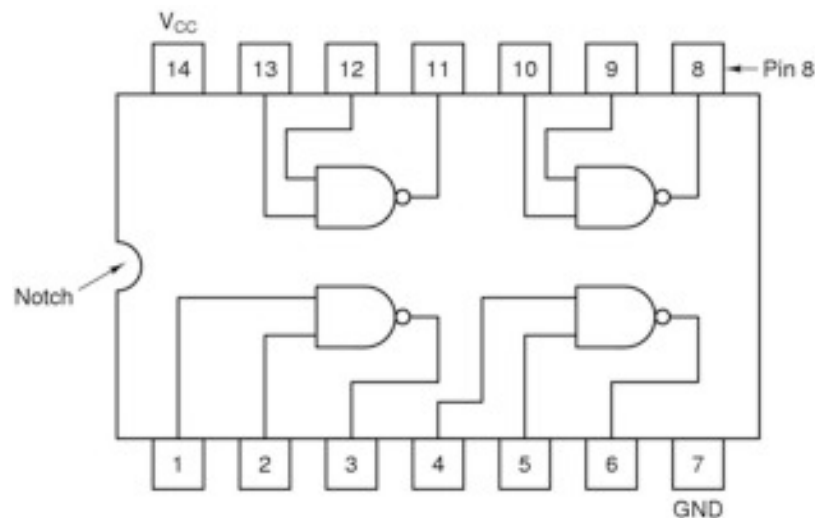
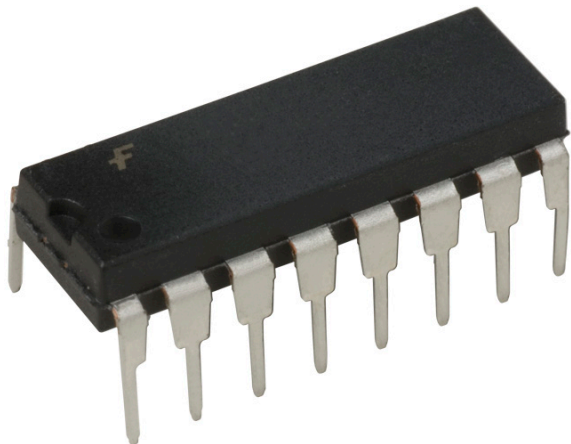


A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0



## Step 2 cont.: Translate truth table into circuit

- These gates are available in integrated circuits that you can buy at electronics stores (RadioShack, Fry's, etc.)



## Step 2 cont.: Translate truth table into circuit

- There are a couple different ways to translate a truth table into a physical circuit.
- One easy way is the Sum of Products method (SOP), another way is the Product of Sums (POS) method.
- These methods create a working circuit that is NOT optimized.
- Uses Boolean algebra (not the same as regular algebra)
  - $A=B+C$        $A=B \text{ OR } C$
  - $A=B*C$        $A=B \text{ AND } C$
  - $A=BC$        $A=B \text{ AND } C$
  - $A=\overline{B}$        $A=\text{NOT}(B)$

## Sum of Products

- Select the rows that generate a TRUE output, and then combine the terms with an OR gate.
- You do this separately for each output value (sw1. . .sw4).

IN2	IN1	SW1	SW2	SW3	SW4	Function
0	0	0	0	0	0	Coast
0	1	1	0	0	1	Forward
1	0	0	1	1	0	Reverse
1	1	1	1	0	0	Brake

IN2	IN1	SW1
0	0	0
0	1	1
1	0	0
1	1	1

SW1=

$$\overline{\text{IN2}} \cdot \text{IN1} + \text{IN2} \cdot \text{IN1}$$

IN2	IN1	SW2
0	0	0
0	1	0
1	0	1
1	1	1

SW2=

$$\text{IN2} \cdot \overline{\text{IN1}} + \text{IN2} \cdot \text{IN1}$$

IN2	IN1	SW3
0	0	0
0	1	0
1	0	1
1	1	0

SW3=

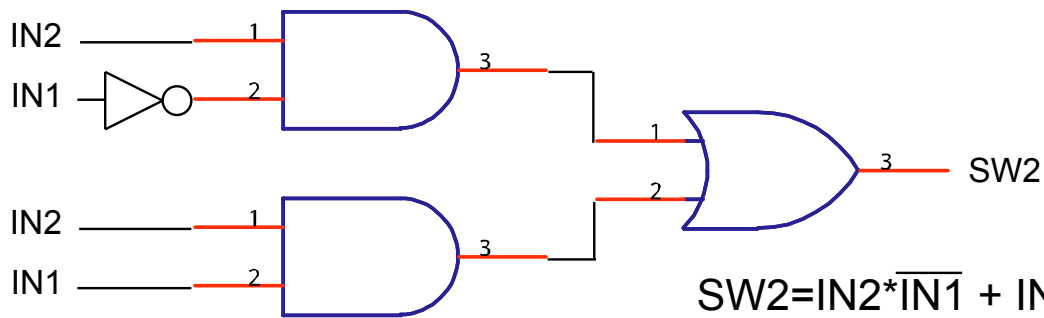
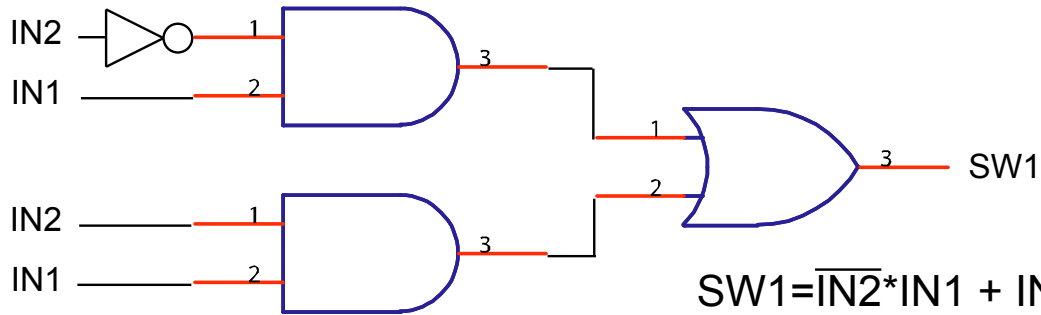
$$\text{IN2} \cdot \overline{\text{IN1}}$$

IN2	IN1	SW4
0	0	0
0	1	1
1	0	0
1	1	0

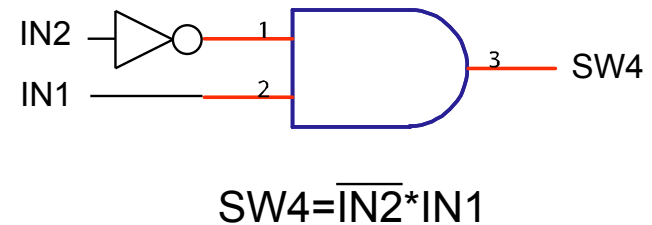
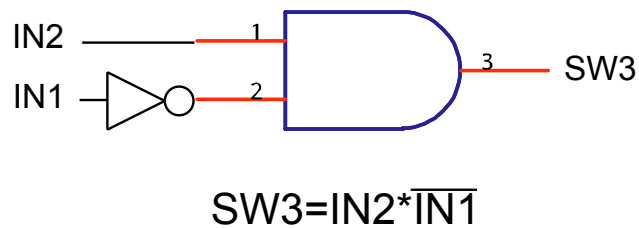
SW4=

$$\overline{\text{IN2}} \cdot \text{IN1}$$

# Sum of Products



Do you see any unnecessary gates?



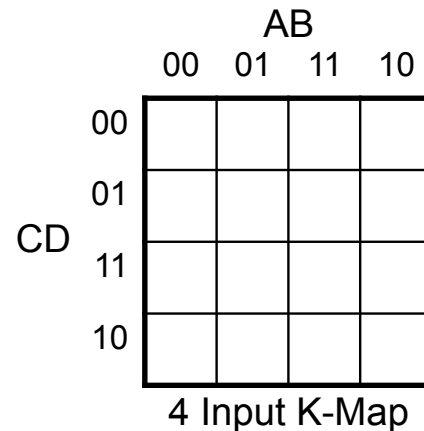
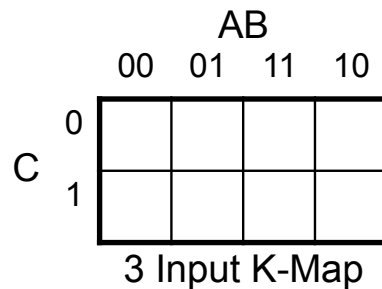
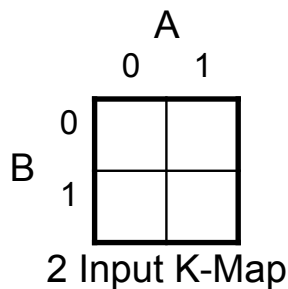
## Step 3: Optimization – Boolean Identities

- Using Boolean identities, the circuit can be simplified.

1.	Law of Identity	$A = A$ $\overline{\overline{A}} = A$
2.	Commutative Law	$A \cdot B = B \cdot A$ $A + B = B + A$
3.	Associative Law	$A \cdot (B \cdot C) = A \cdot B \cdot C$ $A + (B + C) = A + B + C$
4.	Idempotent Law	$A \cdot A = A$ $A + A = A$
5.	Double Negative Law	$\overline{\overline{A}} = A$
6.	Complementary Law	$A \cdot \overline{A} = 0$ $A + \overline{A} = 1$
7.	Law of Intersection	$A \cdot 1 = A$ $A \cdot 0 = 0$
8.	Law of Union	$A + 1 = 1$ $A + 0 = A$
9.	DeMorgan's Theorem	$\overline{AB} = \overline{A} + \overline{B}$ $\overline{A + B} = \overline{A} \cdot \overline{B}$
10.	Distributive Law	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ $A + (BC) = (A + B) \cdot (A + C)$
11.	Law of Absorption	$A \cdot (A + B) = A$ $A + (AB) = A$
12.	Law of Common Identities	$A \cdot (\overline{A} + B) = AB$ $A + (\overline{A}B) = A + B$

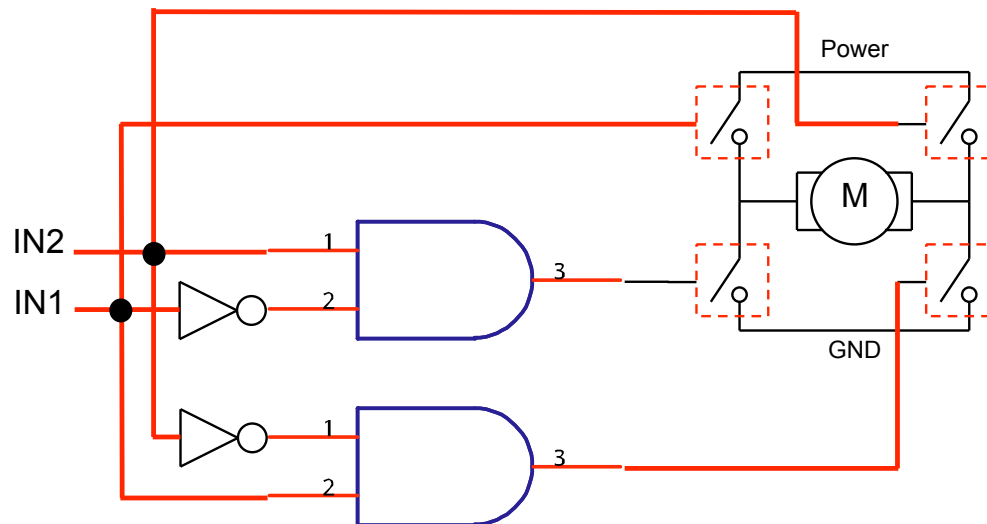
## Step 3: Optimization – K Maps

- Karnaugh Maps are a graphical way to optimize circuits.
- It involves populating the K-map tables from the truth tables with the correct values, and then grouping rectangles of 1's together according to certain rules.



## Step 4: Build it!

# Final Circuit



## Conclusion

- This method is great for prototyping because
  - 1) It is really cheap. Each discrete logic chip is about 10 cents.
  - 2) It is reliable, the chips do what you want them to do as soon as you get them. No programming necessary.
- The catch: it does not scale well for anything more than a simple circuit.
- FPGA (field programmable gate arrays) are used for more complex circuits, and require the user to program the chips.